0

RAND

MapView User's Guide

Larry McDonough, Scott Bailey, Allison Koehler

Project AIR FORCE
Arroyo Center

The research reported here was sponsored by the United States Air Force under Contract F49620-91-C-0003; and by the United States Army under Contract No. MDA903-91-C-0006. Further information may be obtained from the Strategic Planning Division, Directorate of Plans, Hq USAF.

Library of Congress Cataloging in Publication Data

McDonough, Larry, 1962–
MapView user's guide / Larry McDonough, Scott Bailey, Allison Koehler.
p. cm.
"Prepared for the United States Air Force and Army."
"MR-160-AF/A."
Includes bibliographical references and index.
ISBN 0-8330-1363-7
1. MapView. 2. Computer war games. 3. United States—Defenses—Computer simulation. I. Bailey, Scott, 1948– . II. Koehler, Allison, 1967– . III. United States. Air Force. IV. United States. Army. V. Title.
U310.M33 1993
793.9'2—dc20 93-10349
CIP

Published 1993 by RAND
1700 Main Street, P.O. Box 2138, Santa Monica, CA 90407-2138
To obtain information about RAND studies or to order documents, call Distribution Services, (310) 393-0411, extension 6686

RAND

MapView User's Guide

Larry McDonough, Scott Bailey,
Allison Koehler

Prepared for the
United States Air Force
United States Army

Project AIR FORCE
Arroyo Center

PREFACE

This document is a preliminary working draft describing software still in development and should be considered an interim reference manual.

MapView was developed under the TLC/NLC (Theater Level Campaign/ Non-Linear Combat) project—a next-generation operational, theater-level model with supporting tools. To our knowledge, no preexisting tools fully satisfied the following requirements:

- Display simulation (or database) objects on a map
- Attach relevant data to the objects
- Modify those data (including position, color, etc.)
- Communicate easily with the model.

Consequently, MapView was developed as a flexible, general-purpose graphics interface to display, manipulate, and query objects on a map. Eventually, MapView will replace the RAND Strategy Assessment System (RSAS) maptool and be incorporated within the Cartographic Analysis and Geographic Information System (CAGIS) environment.

This work was sponsored jointly under the Theater Force Employment Program of Project AIR FORCE and under the Army Research Division's Arroyo Center. Project AIR FORCE and the Arroyo Center are two of RAND's federally funded research and development centers.

Inquiries and comments are welcome. They may be sent directly to the authors or to Dr. Richard Hillestad, Senior Researcher and Project Leader for the TLC/NLC project.

CONTENTS

FIGURES

ACKNOWLEDGMENTS

Several members of the TLC/NLC (Theater Level Campaign/Non-Linear Combat) project development team and the MOSF (Military Operations Simulation Facility) programming staff have made valuable suggestions regarding the structure and syntax of GOAL (Graphic Object Animation Language) and ease of use of MapView's user interface. They include Richard Hillestad, Louis Moore, Corinne Replogle, Greg Daniels, and Robert Weissler.

1. INTRODUCTION

MapView is a general-purpose, object-oriented graphics program that was developed as part of the Theater Level Campaign/Non-Linear Combat (TLC/NLC) project at RAND. Every attempt has been made to generalize the functionality of the program for use by other projects. MapView is written in the C programming language and runs under the X11 Release 5 windowing environment with Sun Microsystems' OPEN LOOK Toolkit. The program enables scenario generation through a flexible, user-friendly interface that defines graphic objects, places them on an underlying image, and modifies or queries them as desired. In addition, MapView can process a file of commands that define and modify graphic objects and create animated simulation output.

The image formats currently recognized by MapView include Cartographic Analysis and Geographic Information System (CAGIS) terrain and features images, Sun rasterfiles (including screendumps, scanned images, and images drawn using MapView), *no* image at all, and vector images rendered using the WDB2 world database.

The predominant feature of MapView is its ability to define custom graphic objects, attach relevant data, and provide an easy way of visualizing and modifying the data. In this function, MapView has proven useful in checking database validity, generating scenarios (preprocessing), constructing runtime animation frames, and for post-processing analysis.

2. REQUIREMENTS AND ENVIRONMENT

Only the following systems and software are required to run MapView:

- X11 Release 5 (R5)
- Color Sun workstations (3, 4, and Sparcstations 1, 2, and 10)
- Sun Microsystems' OPEN LOOK Toolkit version 3 or higher.

MapView does, however, recognize a few UNIX environment variables if they exist and are set. These variables and their definitions are listed below.

MAPVIEWBITMAPPATH	specifies a list of additional directories that are to be searched when looking for a particular bitmap file. The format of this variable and the following PATH variables is the same as the standard UNIX PATH variable: a list of directories separated by colons (:).
MAPVIEWGOALPATH	specifies a list of additional directories that are to be searched when looking for a particular Graphic Object Animation Language (GOAL) file.
MAPVIEWIMAGEPATH	specifies a list of additional directories that are to be searched when looking for a particular image file.
MAPVIEWHELPDIR	specifies a directory for storing MapView on-line help files.

3. USING THIS GUIDE

Learning to use MapView is a relatively simple process. It requires that the user become familiar with the basic features of MapView's interface. The user interface conforms to Sun Microsystems' OPEN LOOK User Interface Guidelines.[1]

The following conventions are used in this manual:

- Terms in **boldface** type are either GOAL commands or portions of GOAL commands, such as object attributes or punctuation characters. Section 6, "Graphic Object Animation Language (GOAL)," describes all the commands and gives examples of their use.

- Terms that are underlined represent text that appears on the screen in interface objects such as buttons, menus, and window labels.

- Examples of GOAL commands and language syntax will be contained within a box.

This guide is organized as follows:

- Section 4, "MapView Basics," describes the basic MapView environment, image area, and control panel. It details all the features of MapView's interface.

- Section 5, "Tutorial," provides a detailed walk-through of MapView's various functions. After completing the tutorial, users will be familiar with most of the major object-definition, -creation, and -display functions available.

[1]See: *OPEN LOOK Graphical User Interface Application Style Guidelines,* Sun Microsystems, Inc., Addison-Wesley, 1990.

- Section 6, "Graphic Object Animation Language (GOAL)," describes GOAL, the language used by MapView. GOAL is a simple, command-oriented graphics language (or "meta-language") that allows MapView to interface with various models and databases through the use of files. Future versions of MapView will communicate directly with other programs via Sun Microsystems' ToolTalk interface. Communication between MapView and the Map Server Interface program (when rendering vector maps) uses ToolTalk. This section also provides definitions and attributes of the terms used in this manual.

4. MAPVIEW BASICS

This section describes each of the windows and menus that make up MapView's user interface. Section 5, "Tutorial," provides a step-by-step walk-through of a typical MapView session. In addition, Section 5 contains instructions for starting up MapView.

Once you have entered the MapView program, you will see one large window and one small window. In this document, the large window will be referred to as the Map Window. The Map Window is where all the drawing and displaying of graphic objects will be done. The small window, labeled MapView Control Panel, contains the buttons Images, Objects, Help, and Exit. The first three buttons are menu buttons, i.e., buttons that display a pull-down menu when they are selected. These pull-down menus lead to other dialogs for loading and saving images and GOAL files, editing objects, drawing, etc., and are discussed in more detail in Section 5, "Tutorial." Also displayed in the control panel are the x and y and corresponding longitude and latitude coordinates of the mouse cursor when it is located within the Map Window.

Throughout this manual, LMB, RMB, and MMB will be used as abbreviations of the Left, Right, and Middle Mouse Buttons, respectively. Unless otherwise specified, references to "clicking", "selecting", or "choosing", objects with the mouse will refer to the Left Mouse Button.

MOVING AND RESIZING WINDOWS

Modifications to the windows conform to the standard X11R5 conventions:

To move the windows around on the screen, point the arrow cursor at the top border of the window and click down (and hold down) the LMB. Position the window on the screen as desired and release the mouse button. Affecting the position or shape of an object while the mouse button is held down is often referred to as "dragging" the object.

To resize the windows on the screen, position the arrow on any corner of the window. Then, with the LMB, drag the corner to the desired size and release the mouse button. NOTE: The size of the Map Window cannot be made any larger than the image that it contains. It can be made smaller, in which case the scroll bars become active for moving around within the image area.

IMAGES MENU BUTTON

The Images menu button is the first of four buttons on the MapView Control Panel. If you click down using the RMB on this button while still holding down the mouse button, a pop-up menu with the following options will appear: Load Image..., Load Colormap..., Erase Image, and Save to Rasterfile. To select any of these suboptions, slide the mouse down the pop-up menu until the desired option is highlighted, and release the mouse button. Each of these options is discussed in the following subsections.

Load Image

Selecting the Load Image option causes a window to be displayed that contains a scrolling list of image files. You may scroll through the available image list by clicking on the scroll bar at the side of the image window. To select an image, click on the name of the desired image, then select the Load Image button in the lower half of the window. MapView will display a "loading image" message in the lower border of the Map Window. In addition, as the image file is being read, a progress window will appear, showing how much of the file has been processed (depending on the size of the image, this can take a few seconds).

Load Colormap

Rarely will colormap files need to be loaded explicitly. MapView automatically opens and reads the colormap file corresponding to the selected image when the image is loaded. The process of loading a colormap file is the same as loading an image file. A colormap file is a regular text file that contains a list of colors (Red-Green-Blue [RGB] values) that are to be used to display the corresponding image. By default, a colormap file has the same name as its image file, except that the filename extension is ".cms" as opposed to ".image".

Currently, only CAGIS images require colormap files; all other image file formats contain the colormap information within the image data file.

Erase Image

The Erase Image option is useful when you want to view the objects over a blank, or single-colored, background. To erase an image from the map area, select the Erase Image button. An "erasing image" message will appear in the Map Window. MapView will erase the currently displayed image and "color" the map area with the background color (default is black). The background color may be specified on the command line to MapView with the "-bg" flag (e.g., mapview -bg MediumSeaGreen).

Save to Rasterfile

This option enables the user to save whatever is displayed in the Map Window to a rasterfile so that it may be loaded as an image later, printed on a laser printer, or imported as a frame in an Apple QuickTime movie.[2] To save the Map Window as a rasterfile, select the Save to Rasterfile button from the Images menu. Enter a filename followed by .ras. Two options (or views) are available when saving the screen to a rasterfile: Write current view or Write whole view. A *whole view* is defined as the whole image (including any portion that might not be visible in the Map Window because of the position of the scroll bars). The *current view* is defined to be that portion of the image that is currently displayed on the screen. If you want to write out the same small view of an image, position the window to contain the desired view and select Write current view. To do so, programmatically within GOAL, see the command "**SnapShot**" in Section 6, "Graphic Object Animation Language (GOAL)."

[2]In practice, saving one frame at a time interactively is just not feasible. See the GOAL command **SnapShot** for more information about producing successive frames of animation.

OBJECTS MENU BUTTON

The Objects menu button on the Control Panel has the following sub-options: Define Types..., Create (Draw)..., Load file..., Save file..., Show by type..., Select/Edit..., Connect.... Selecting any of these menu buttons will display a corresponding window that enables the user to manipulate the graphic objects in various ways. These menu options are discussed in the following subsections.

Define Types

The Define Types... option allows the user to define new graphic types (sometimes called "classes" or "templates"), so that objects can be customized for a particular model, database, or study. Defining new object templates is also useful in controlling several groups of objects at one time (for instance, displaying or erasing all objects of a certain type). New object templates are defined by adding attributes to the existing "base types" offered by MapView or to previously user-defined types. Once a new type is defined, it may then be a base type for future object definitions.

Open the Define New Types window by selecting Define Types... from the Objects button in the main control panel. Type in the name of the new object class on the Type Name: line and choose a base graphic type from the scrolling list located in the top left-hand corner of the window (labeled Base Types). To add an attribute to this new definition, type the name of the attribute on the Attribute Name: line and select the attribute's data type from the scrolling Attribute Types: window by clicking on its name (Integer, Float Array, etc.). Select the Add Attribute button and notice that your new object definition, shown in the New Type Definition: scrollable window, contains the new attribute definition. If you make a mistake while defining a type or adding an attribute, you may use the Delete Attribute, Delete Type, or Clear All buttons to clear the error. When satisfied with the input for the new object type, click the Define This Type button. The new object type, once defined, will be added to MapView's Create (Draw)... list and may now be used as a template for creating instances of this object type. This new type will also be added to the Base Types list of this dialog so that it may be used as a base class for further hierarchical definitions.

Create (Draw)...

Another option under the Objects pop-up menu is the Create(Draw)... button. Before objects can be created in the Map Window, the user must select the type of object to draw from the scrolling list labeled Select Object type to draw:. Select the desired object type to create and notice that the appropriate graphics attributes are highlighted and become available in the window below. Attributes that do not make sense for a particular object type (e.g., the **font** attribute for a line segment object) will be grayed-out. If the object being drawn is not named, then MapView will give the object a unique name. All graphic objects must have a unique name. The default names given to objects are composed of two parts separated by an underscore. The first part is the object's type (or class), and the second part is a unique number calculated by MapView (for example: NetworkRegion_42). The attributes that are active for a given graphic type will be displayed in **bold-faced** type on the screen. Some of the attributes have corresponding buttons that bring up scrolling lists of choices: Colors..., Patterns..., Bitmaps..., Fonts.... Selecting a value in a list causes that value to be reflected in the attribute for that object. By using the selection lists, you can set the values of the attributes (color, font, line width, pattern, etc.) by using the mouse, *not* the keyboard.

To draw the object once its attributes are set, select the Draw button, then move the arrow into the map area. Once inside the map area, the arrow cursor changes into a crosshair. Position the cursor at the desired drawing location and click the mouse appropriately. Refer to Section 5, "Tutorial," for specifics about how to draw the different types of graphic objects. The process of drawing each object differs slightly because the objects are different; as a whole, the drawing process resembles similar drawing programs, such as MacDraw. To exit the drawing mode, click the Done button. As with most pop-up windows in MapView, the window closes when you click the Dismiss button.

Load file

To load a GOAL file, select the Load file... option from the Objects button. The Load Objects window will appear with a scrolling list containing all the GOAL files in the current MAPVIEWGOALPATH environment variable. By default, the MAPVIEWGOALPATH variable contains ".", so that any files in your current directory will be displayed in the list. Select the desired filename (which becomes highlighted and appears on the Overlays file: line). Load the file by clicking the Load Button, and dismiss the Load Objects window by clicking the Dismiss button.

Save file

To save the current set of graphic objects and definitions to a GOAL file for later retrieval or to send them to a model or other program, select the Save file... option from the MapView Control Panel's Objects menu button. Type in a filename (usually ending in ".goal") or select one from the list provided. Note, however, that saving objects to an already-existing file will overwrite the contents of that file. If you try to do this, MapView will prompt you with a message to make sure that you want to overwrite the file in question.

Show by type

The Show by type window allows the user to display or undisplay graphic objects according to their type. It is useful for uncluttering the screen when many objects are displayed. To open the Show/UnShow Objects by Type window, select the Show by type... button from the MapView Control Panel's Objects menu button. The window that appears contains a scrolling list of all the currently defined object types. The objects that are depressed (or highlighted) are currently being shown and are the default state for all objects. To unshow a class of objects, just click the object type (thus unhighlighting it). There are two convenience buttons for showing or unshowing all the graphic types: Show All Graphic Types and UnShow All Graphic Types. Note that "unshowing" an object is not the same as destroying one. Unshown objects may be reshown, whereas a destroyed object must be re-created.

Select and Edit...

The Select and Edit... window provides a variety of options for selecting, viewing, and editing objects and their attributes. Open this window by selecting the Select and Edit... menu option from the Objects menu button off the Main Control Panel. The options in this window are relevant to all the selected objects. To select an object, click the LMB near one of its edges. To select more than one object, hold down the shift key and select other objects. If the objects are relatively close together, you may select more than one object by clicking down with the LMB and dragging a box around the desired objects. To unselect an object, just select it again. Once an object, or set of objects, is selected, the following options are available from this dialog window: Identify, Show, Front, Select All, Edit, UnShow, Back, UnSelect, and Destroy. There are also four arrow buttons that may be used to move the selected objects around on screen. The number of pixels that an object moves with each click on one of the arrow buttons is specified by the value in the Incr item. Selected objects may also be moved by dragging them around with the LMB from within one of their tagged corners.

Connect

The Connect button is used to connect MapView directly to other input streams that are presumably connected to other processes. Currently, this option is only partially implemented. The only option available to the user is connecting to MapView's standard input stream. Click the Standard input button to connect to standard input. MapView will now be expecting to execute GOAL commands from the window that it was executed from (probably an xterm or shelltool). If you click this button inadvertently, you may exit this mode by typing **EOF** followed by a Return in the terminal window that MapView was started from.

HELP BUTTON

The Help button has three suboptions: General Help..., GOAL SYNTAX..., and Bug Reporter. The first two options open windows that contain the contents of the relevant help files. If the help files are not available or cannot be found, the windows will display an error message. The Bug Reporter button opens a dialog window that allows the user to submit a bug or comment about MapView directly to the author via email. This option works only if your system supports UNIX email. NOTE: If you are not logged in at RAND, your system must also have access to the Internet in order for this option to work properly. For more clarification about electronic mail and the Internet, see your system administrator. Comments are always welcome by phone.

EXIT BUTTON

The Exit button quits MapView. You will be prompted with a message such as: **"Do you really want to exit MapView?"** If you exit and have not saved your work, your objects and their definitions will be lost. NOTE: MapView does not save graphic objects and their definitions automatically. If you want to save your work, you must use the Save to file option under the Objects menu from the MapView Control Panel.

5. TUTORIAL

The previous section outlined the various windows and menus that compose MapView's user interface. This section walks you through a typical MapView session (loading images, creating graphic objects, defining new object types, etc.). It is recommended that you read this section while running MapView. NOTE: This tutorial does not cover all the menus in MapView. See Section 4, "MapView Basics," for complete coverage of menus and options.

START UP MAPVIEW

To run MapView, make sure that the executable (mapview) is in your search path and that you are running Sun X11R5 version 3 or higher.

Starting up X11R5 may differ from site to site; usually, typing openwin once you have logged in will work. If this does not work, check with your site administrator for specifics. If you are running MapView at RAND from within the MOSF (Military Operations Simulation Facility), MapView will already be in your path and you may skip the remainder of this paragraph. The only other runtime environment setting necessary to run MapView is access to the standard X11R5 dynamic libraries.[3] Again, this is already set up for you in the MOSF.

Start up MapView by typing mapview at your UNIX prompt (in this example, "tutorial%" is the prompt):

```
tutorial% mapview <CR>
```

[3]In UNIX, you may get a list of the dynamic libraries that a program depends on by using the ldd command (for example: ldd mapview). Although the directory paths of these libraries may be added to the LD_LIBRARY_PATH environment variable, it is more desirable to include the path information at link time with the -L option to cc.

If you get the statement "MapView: Command not found", check your environment variables and search path and make sure that the MapView program is installed on your machine, and that you have access to the executable file.

If you get the statement "Cannot execute binary file. MapView: Exec format error", verify the compatibility of your machine with the version of MapView. For example, you may be attempting to execute a Sun 3 binary on a Sun 4 machine.

Once MapView is running, you should see two windows on your screen that resemble Figure 5.1. The smaller of the two windows is the MapView Control Panel. The four buttons in this window provide the menus that initiate most of MapView's functions. The Images menu button lists all options for dealing with images.

The larger window is the graphics window (also called the map area or Map Window). All images, overlays, and drawing will take place in this window.

LOAD AN IMAGE

Using the Images menu button from the control panel, select Load Image.... Remember that to get a menu button's pop-up menu to display, you must click down with the Right Mouse Button and not release the mouse. Slide the mouse down and release it over the desired option (in this case, Load Image...), then release the mouse button. When this is done, another window appears with a scrolling list of available images. Select the image "Tutorial.image" by clicking on the name in the list. Once selected, the name of the image will appear in the window.

Clicking on the Load File button will cause MapView to load the image file and display it in the large window, as shown in Figure 5.2.

Clicking on the Dismiss button in the Load Image window will cause the small Load Image dialog window to disappear but will not affect the display in the large window.

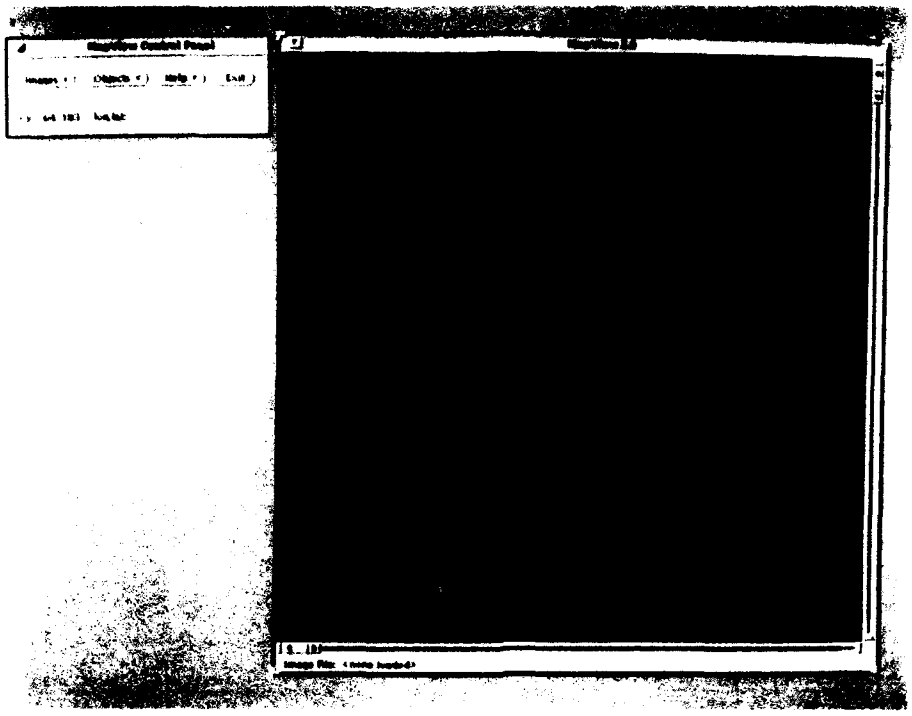


Figure 5.1—Initial MapView Display

Figure 5.2—Load Image File Dialog

LOAD AN OVERLAY (GOAL) FILE

After pressing the Objects menu button from the MapView Control Panel, select Load file.... Another window appears with a scrolling list of available overlay files. Select the file "Tutorial.goal" by clicking on the name in the list. The name of the file will now appear in the window as shown in Figure 5.3.

Clicking on the Load button will instruct MapView to load the overlay file and display the objects on top of the image in the large Map Window.

Clicking on the Dismiss button in the Load Objects window will cause the small Load Objects dialog window to disappear but will not affect the display in the large window.

CREATE POLYLINE OBJECTS

After pressing the Objects menu button on the Control Panel, select Create (Draw). Another window will appear with a scrolling list of available object types, as shown in Figure 5.4. Select the Polyline object type by clicking on its name in the list. Change the line width to 4 by clicking four times on the "up arrow" button next to the line width: number. To select a color for the object, either type in the name of the color on the color: line or click on the button labeled Colors... to display a scrolling list of colors to choose from. Remember, the options that are available are in **bold-faced** text; those that are not available are grayed out. Notice, also, that a default object name has been created for this object already and is displayed on the Object's Name: line in the dialog window.

Once the desired drawing options have been selected, click on the Draw button. The appearance of the Create Graphic Objects window changes to show the new object's name and to provide a reminder to <<Select "Done" when finished>>.

Now move the cursor into the large display window and notice that its shape has changed to a crosshair. The cursor will always appear this way when you are in drawing mode. Click where you would like to begin the polyline. Move the crosshair to another point of your choice, and click again. Continue doing so a few more times. When finished, click the Done button to complete the object-creation process.

Figure 5.3—Loading a GOAL File

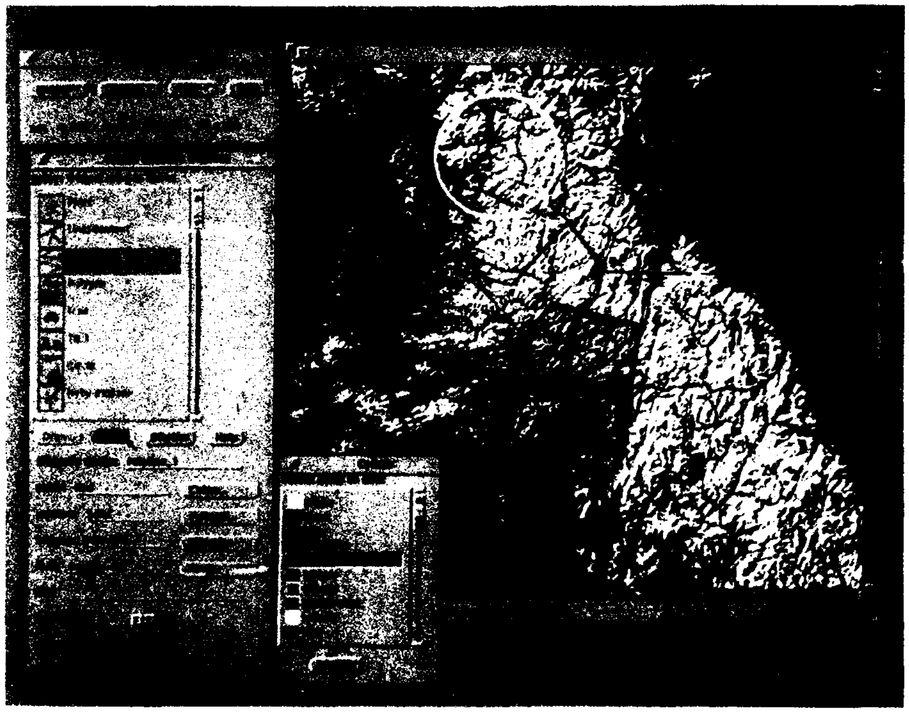


Figure 5.4—Creating Graphic Objects

View and Modify the Polyline's Attributes

In order to view the attributes of an object, the object must first be selected. To select an object, click the LMB near the edge of the object. The selected object will be displayed with small red squares at each vertex and end point. The color of the selected objects will also change to the "foreground" color as specified on the command line to MapView (e.g., mapview -fg red). The default foreground color is white.

Using the Objects menu button from the Control Panel, select Select and Edit.... The Select and Edit dialog will appear as shown in Figure 5.5.

From this dialog window, select the first option under the Identify menu button labeled Last Selected. In this case, we are interested in displaying the attributes of the most recently selected object as opposed to all the selected objects. A window will appear that displays the attributes (and their corresponding values) of the polyline object that you just created, as shown in Figure 5.6.

After examining the attributes, dismiss the Identify window (Polyline_4 window in Figure 5.6), but not the Select and Edit window (you will use this window in the next subsection).

Move the Polyline

The four arrow buttons at the right of the Select and Edit window are used to move objects that have been selected. If no objects have been selected, these buttons have no effect. The distance the object will be moved is determined by the "Incr" field in the dialog window. You can adjust this increment using the small up-down arrows or by typing a new increment (in pixels) on the line. Set the increment for any number up to 50 and move the object in any direction, using the large arrow buttons. Another way to move an object is to drag it by one of its selection boxes: Click down (with the LMB) inside one of the red selection boxes of the object you want to move, drag the mouse to the desired location, and, once there, release the mouse button.

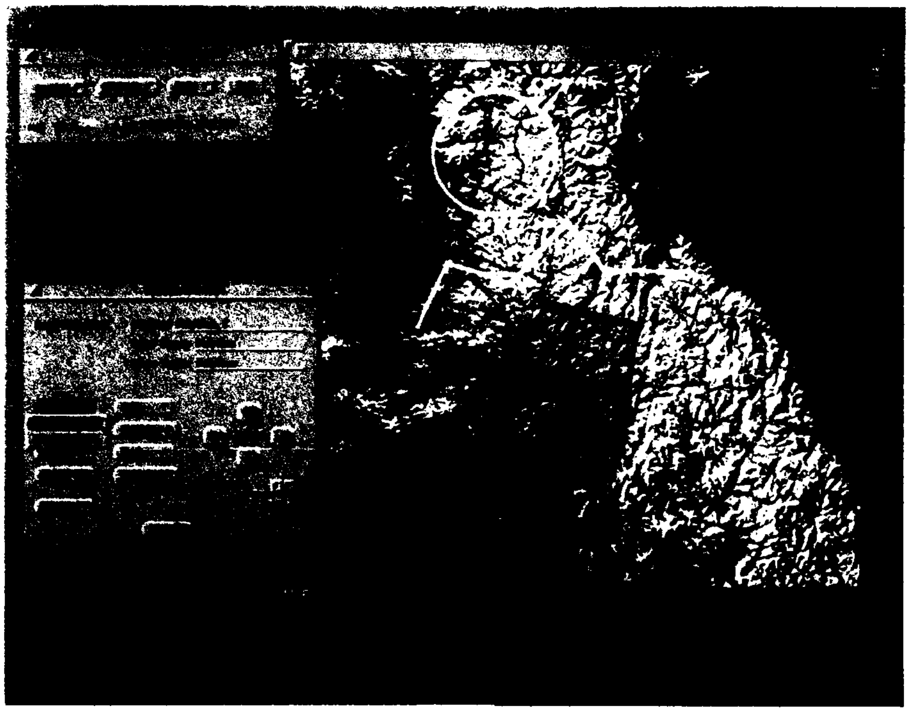
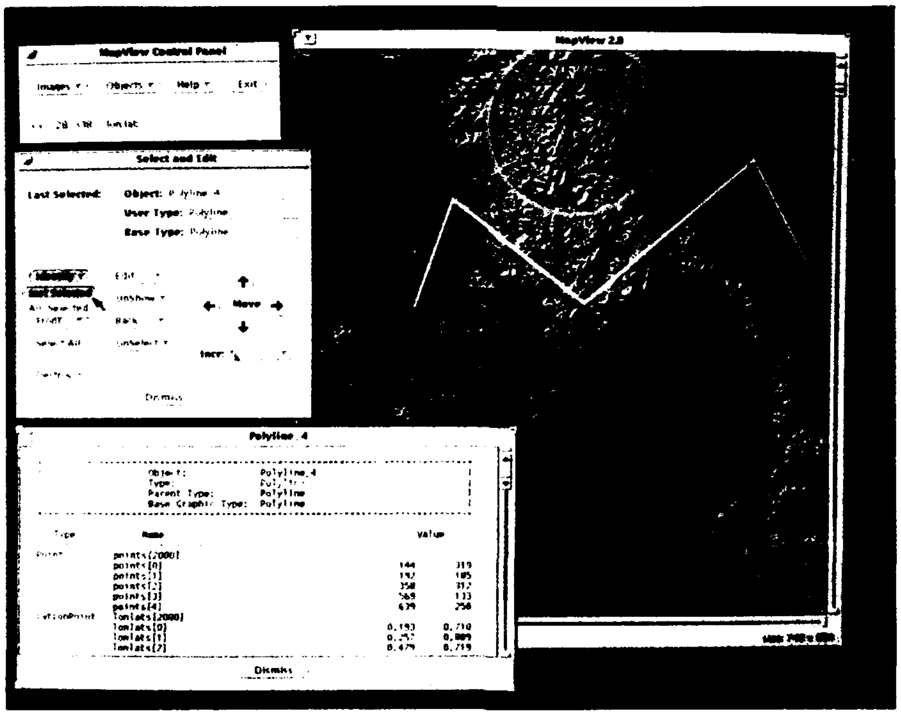


Figure 5.5—Modifying Selected Objects



Last Selected: Object: Polyline 4
User Type: Polyline
Base Type: Polyline

Polyline_4

```
Object:              Polyline_4
Type:                Polyline
Parent Type:         Polyline
Base Graphic Type:   Polyline

Type          Name                         Value
Point         points[2000]
              points[0]                    144      319
              points[1]                    192      185
              points[2]                    358      312
              points[3]                    569      133
              points[4]                    639      256
[illegible]   lonlats[2000]
              lonlats[0]                   0.193    0.710
              lonlats[1]                   0.257    0.809
              lonlats[2]                   0.479    0.719
```

Figure 5.6—Viewing an Object's Attributes

Unshow and Reshow the Polyline

To erase the polyline object from the screen, use the left mouse button and click on the UnShow menu button. The object has not been destroyed, just moved from the display list to an off-screen hold list. Since we used the left mouse button instead of the right one on a menu button, the default menu item (the first) was selected. To reshow the object, use the left mouse button and select the Show menu button. The object is now back in the display list and visible on the screen.

Destroy the Polyline

To destroy the selected polyline object, use the Destroy button in the same manner as the Show and UnShow buttons above. Once an object is destroyed, it no longer exists. Dismiss the Select and Edit window.

CREATE ICON OBJECTS

An Icon object is represented graphically by its bitmap. A bitmap is a two-dimensional array of binary pixels in which each pixel is either "on" or "off". An Icon's bitmap attribute is just one of the attributes that define an Icon object. Icon objects have other attributes such as **x** and **y** that specify its location and **color** which specifies what color should be used to render the "on" pixels in the bitmap.

An Icon object can be created by going back to the Create Graphic Objects window and selecting Icon. The appropriate attributes for an Icon object are now active (not grayed out). They include **name**, **color**, and **bitmap**, and all three attributes have initial default values. Change the icon's name to Star (this may be done by deleting the name that is shown and typing in a new one). Using the left mouse button, click on the Colors... button to display a window of available colors. A Colors dialog like that in Figure 5.4 will be shown.

Using the scroll bar in the Colors window, scroll down through the list until the color of your choice is visible. Select it by clicking on the color in the scrolling list (LMB). Note that the color you have selected now appears in the Create Graphic Objects window. Dismiss the Colors window and select the Bitmaps... button in the Create Graphic Objects window. Various bitmaps are available in the Bitmaps scrolling list that appears as shown in Figure 5.7. Several are default bitmaps that are present whenever MapView is invoked. The additional bitmaps were defined and created in the Tutorial.goal overlay file that was loaded earlier. From the Bitmaps window, select FBigStar. Notice that its name has replaced the original bitmap name in the Create Graphic Objects window.

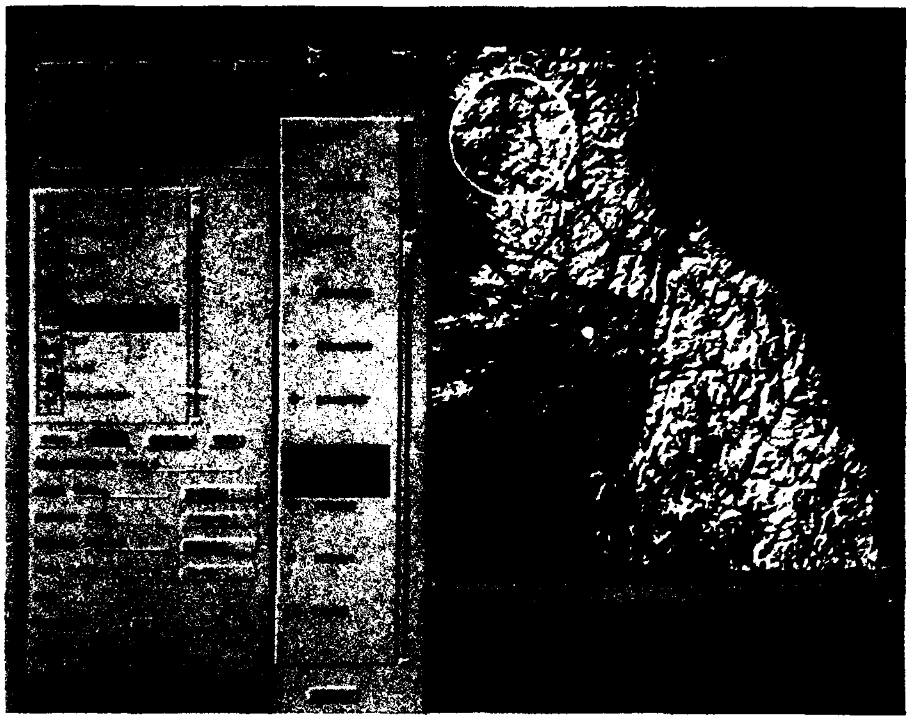


Figure 5.7–Selecting a Bitmap for an Icon Object

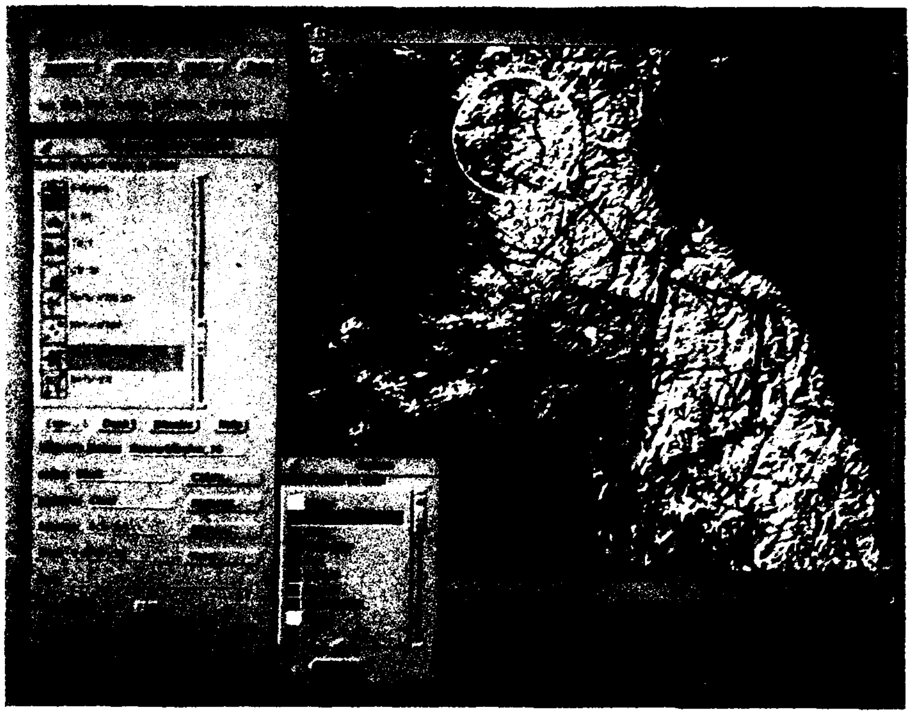


Figure 5.8–Drawing a NetworkRegion

Place several stars in the large display window by first clicking on Draw and then clicking inside the Map Window. New object bitmaps and colors may be changed at any time during the drawing process. When finished, select Done from the Create Graphic Objects window.

CREATE NETWORKNODE OBJECTS

NetworkNode objects are drawn the same and look the same as Icon objects, except that NetworkNode objects can be used as end points in defining NetworkArcs and as vertices when defining NetworkRegions.

Using the Create Graphic Objects window, select NetworkNode. The appropriate attributes for a NetworkNode object are now highlighted and active. They are the same as for an Icon object. Change the bitmap attribute to FCircle2 and draw some NetworkNodes using the same process you used when drawing the Icon objects. When finished, select Done from the Create Graphic Objects window.

CREATE NETWORKARC OBJECTS

Now select NetworkArc from the same window. The NetworkArc attributes (color and line width) are now active. Select a color and line width of your choice and click Draw.

Draw NetworkArcs

NetworkArc objects are defined by their two end nodes (NetworkNodes). To draw a NetworkArc object, click (LMB) close to a NetworkNode, thereby fixing one end of the NetworkArc. Notice that a line tracks the cursor movement. To finish drawing the arc, click close to the NetworkNode that will correspond to the end of the arc. If both nodes are the same, no arc object is created. Repeat this process as often as you like. Click Done when appropriate.

NOTE: Since NetworkArcs are defined by their bounding end nodes, they cannot be moved around on the screen as can other objects (this is also true for NetworkRegion objects). In order for these objects to be moved or reshaped, their bounding nodes must be moved (the arcs and regions associated with those nodes will automatically snap to the new locations when their nodes are moved).

CREATE NETWORKREGION OBJECTS

NetworkRegion objects are also defined by NetworkNodes. Attributes available for drawing a NetworkRegion object include **color**, **pattern**, and **line width**. Select the pattern of your choice from the Patterns... menu in the same fashion as for bitmaps. Custom patterns (as with bitmaps) may also be added to the list. See the **DefineBitmap** and **DefinePattern** GOAL commands in Section 6 for more information.

Draw a NetworkRegion Object

Draw a NetworkRegion object by clicking on or near the NetworkNodes defining its boundary. Continue to click on nodes around the periphery of the desired region. There is no need to click again on the first node to close the loop. MapView will do this for you when you click Done. Figure 5.8 shows the Create Graphic Objects window and the Color dialog while a NetworkRegion is being drawn.

MOVE A NETWORKNODE

Select one of the NetworkNode objects that bounds one of the arc or region objects that you have drawn. Drag the node to another location by clicking down (and holding) the LMB on one of the selection boxes that surrounds the selected node. When you release the mouse to position the node, notice that the arcs and regions that use that node have reshaped themselves according to its new position. Selected objects may also be moved by using the positional arrows in the Select and Edit window.

DESTROY A NETWORKNODE

Destroying a NetworkNode also destroys any arc referencing that node and removes that node reference from any region. To demonstrate this, destroy one of the NetworkNode objects you just created. Once all the nodes that bound a region have been destroyed, the region object will be destroyed automatically. Note, however, that the reverse is not true. Destroying arcs or regions will never affect any node objects:

Node objects are independent; they are not defined (or bounded) by any other objects. Selected objects may also be destroyed by hitting the delete key.

CREATE CIRCLE OBJECTS

Using the Create Graphic Objects window, select Circle. The appropriate attributes for Circle objects are now active. They include **color, line width**, and **pattern**. Click Draw and place the cursor where you want the center of the circle to be. Click and release the cursor. The outline of a circle will now follow the cursor as it moves away from the center. Click the mouse again when the circle is the desired size. Repeat this process for as many circles as you like, and click Done when finished.

CREATE TEXT OBJECTS

Text objects are handled the same as are all the other graphic objects: When you select Text as the drawing object class, its attributes will become active. Attributes include **color**, **font**, and **value**, a value being the actual lettering of the text object. Select the color and font of your choice, then type in the "value" of your text object on the text line. The text may be edited before or after the object is placed, but it saves time to have it written correctly before initial placement on the display. To place the text, click Draw and place the cursor where you want the text to begin. As the cursor enters the display area, it changes its shape to a crosshair and a text box to show the dimensions of the text. Click and release the cursor and the text will appear. An example of the fonts available for the Text object can be seen in Figure 5.9. As with the other objects, any of the appropriate graphic attributes (in this case, **color**, **font**, and **value**) may be changed at any time during the drawing process. In other words, to enter a second Text object with a different color and font, you do not need to exit the drawing operation. After a few Text objects have been placed on the screen, click Done.

SAVE OBJECTS TO A GOAL FILE

Up to this point, you have created a number of graphic objects and displayed them over an image. To save these objects to a GOAL file, an ASCII file that contains graphics commands, select Save File from the Objects menu button. The Save Objects dialog window will appear. Type in a filename (e.g., "test.goal") or select one from the scrolling list. If the file already exists, MapView will warn you that you are about to overwrite the contents of another file. You will be given a chance to enter a new filename or go ahead and overwrite the existing one.

VIEW THE GOAL FILE

To view the file that you have just created, execute the following UNIX command from another xterm or shelltool window:

```
tutorial% more test.goal    <CR>
```

Notice the commands in this file that correspond to the objects that you have created in addition to those objects that originated in Tutorial.goal. When you are finished, return to the MapView Control Panel and exit MapView.

RELOAD THE GOAL FILE

Restart MapView and load the GOAL file you just saved by selecting the Load File... option from the Objects menu button. The image and objects should all be displayed on the screen just as they were when you exited MapView.

DEFINE A NEW GRAPHIC OBJECT TYPE

To define a new graphic object type, from the Objects menu button on the MapView Control Panel, select Define Types.... The Define New Types window will appear. See Figure 5.10. To derive a new graphic object type from an existing type, first select the base graphic type by clicking on Polyline in the Base Types scrolling list. Notice that the base-type text attribute is filled with Polyline and the Polyline object's attributes are listed in the New Type Definition scrolling list.

To give this new graphic type a name, enter "River" on the Type Name line. Type the word "Depth" on the Attribute Name line. Declare this attribute as type Float by clicking on Float in the Attribute Types scrolling list. Now add this new attribute by clicking the Add Attribute button. Notice that this attribute appears in the New Type Definition scrolling list. To inform MapView of this object type definition, click on the Define This Type button. At this point, our new graphic type, River, appears in the Base Types scrolling list, allowing us to hierarchically define graphic objects. In other words, we can now define a new graphic object type based on the object type River.

CREATE INSTANCES OF THIS NEW TYPE

Using the Objects menu button from the Control Panel, select Create/Draw. Our new type will appear at the bottom of the Create Graphic Objects window. Select the object River by clicking on its name in the list, and draw an object of type River as you would a Polyline.

SAVE THE SCREEN TO RASTERFILE

The entire display may be written out to Sun Microsystem's graphics image file format, rasterfile. To save the display as a rasterfile, dismiss the Create Graphic Objects window and select Save to rasterfile under the Images menu button. The default name is "mapview.ras", but it may be changed to any name of your choice. Change the name now and click Write whole view. Once the image has been saved, exit MapView by clicking the Exit button on the Control Panel and Yes on the resulting confirmation window. Reenter MapView and load the new image that you just saved (see **LOAD AN IMAGE**, p.16). Notice that the objects are actually part of the image now and are no longer selectable. This is an example of how you would augment an existing image with roads, cities, etc.

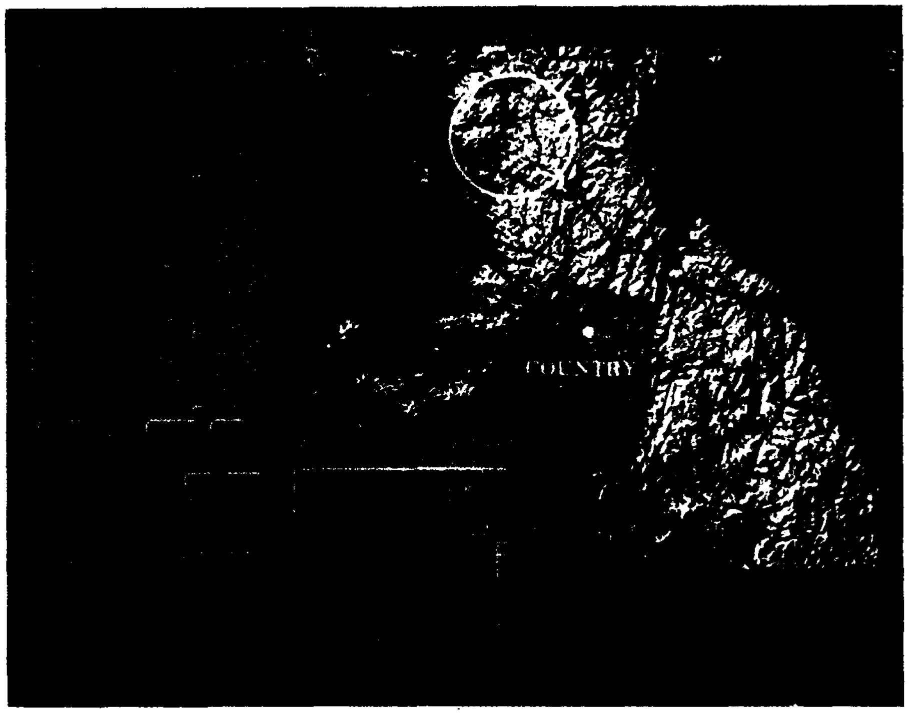
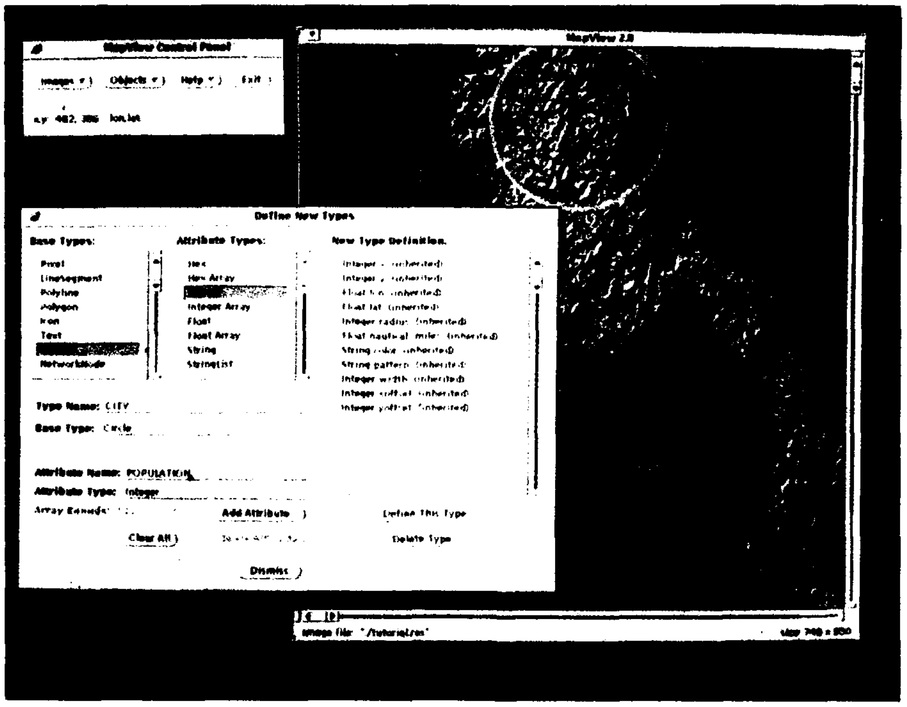


Figure 5.9—Creating Text Objects



Integer x (inherited)
Integer y (inherited)
Float lon (inherited)
Float lat (inherited)
Integer radius (inherited)
Float nautical miles (inherited)
String color (inherited)
String pattern (inherited)
Integer width (inherited)
Integer xoffset (inherited)
Integer yoffset (inherited)

Type Name: CITY
Base Type: Circle
Attribute Name: POPULATION
Attribute Type: Integer
Array Bounds:

image file: "/tutorial.ras"

Figure 5.10—Defining New Graphic Types

6. GRAPHIC OBJECT ANIMATION LANGUAGE (GOAL)

GENERAL STRUCTURE

Graphic Object Animation Language (GOAL) is a simple, command-oriented language. The syntactic structure of GOAL resembles that of C++; however, at present, there are no conditional or looping constructs available in GOAL. All commands begin with a keyword and are optionally followed by a parameter or block of information enclosed in braces "{", "}".

Most GOAL commands affect one of two internal linked lists: the *display list* and the *hold list*. The hold list is used to hold new and unshown objects. As its name implies, the display list contains the objects that are currently displayed on the screen.

GOAL Commands

The following subsections comprise the commands, their descriptions, and examples of their use. The commands are listed in alphabetic order. Examples of GOAL commands will be outlined in boxes below. In addition, the "#" character represents the beginning of a comment in GOAL. In other words, all characters after the "#" up to the end of the line are ignored by the command interpreter.

BackGraphic

The **BackGraphic** command may be used to move an object to the back of the visual stack (i.e., overlapping objects will appear in front of the specified object). This command takes the name of a graphic object (or an object type) as its only parameter. See also **FrontGraphic**. For example:

```
BackGraphic myObject       # just this object moves back
BackGraphic myObjectType   # all objects of this type move back
```

CreateMenu and AttachMenu

MapView's user interface may be extended to permit additional menu items and submenus, or pull-right menus, to be attached to MapView's default menu. The default menu (also called the "Main Menu") pops up when the right mouse button is held down while the cursor is somewhere over the Map Window. As a default, the main menu contains the following three items:

• Show Control Panel	Selecting this option will redisplay the control panel if it was unpinned, dismissed, or hidden
• Refresh	Selecting this option will cause all the graphic objects and the underlying map to be redrawn
• Exit MapView	Selecting this option offers yet another way to exit MapView.

Additional menu items will appear in the main menu *below* the three default menu items listed above. The *actions* taken by the selection of these new menu items can be defined using GOAL or extended by any of countless UNIX utilities, programs, scripting languages, etc., as the following examples make clear.

There are two commands that facilitate creating and using menus. These are the **CreateMenu** command and the **AttachMenu** command. The relationship between these commands is analogous to that between the **CreateGraphic** and **ShowGraphic** command: Menu objects are instantiated by using the **CreateMenu** command, but they are not *displayed* until they are attached to the Main Menu. For example, to create a simple menu with three items in it and attach the menu to the Main Menu you could do the following:

```
CreateMenu "Extra Menu Options"
{
    item = { "xterm", "Run \"xterm -fn screen-10&\"" };
    item = { "shelltool", "Run \"shelltool&\"" };
    item = { "save objects", "SaveToFile save.goal" };
}

AttachMenu "Extra Menu Options"
```

In this example, there are three distinct menu items, each with only one action to be taken upon selection. The **item** attribute in this case is of type **StringList** and is structured in the following way: The first item in the string list is the label of the menu item (which is what the user will see). The second and remaining items in the string list are the commands to be executed upon selection of that item. Commands requiring double quotes (as in the **Run** command) must have their embedded quotes preceded by a backslash so that the language parser knows that this quote does not signify the end of the string.

In the above example, the **Run** command is used to pass the following string onto the UNIX shell. Each of the commands passed to UNIX will be executed in the background (since the "&" was used). Background execution will allow MapView to continue to process the rest of the menu item's commands (if any exist) immediately. If the desired effect is to wait (or synchronize) the actions, then you would not append the "&" at the end of the **Run** commands (MapView will wait until UNIX returns before processing the next action).

To create submenus (sometimes called pull-right, or walking menus) you use the attribute **menu** instead of **item**. For example, let us create a new menu and make the menu we defined above one of its submenus:

```
CreateMenu "My Root Menu"
{
    menu = "Extra Menu Options";
}

AttachMenu "My Root Menu"
```

In this example, we have created a menu with the label "My Root Menu" and given it one pull-right menu, which is the one we defined before. The **menu** attribute is of type **String**. A menu may have as many **item** and **menu** attributes as you like and may be nested to virtually any depth. Also, all pull-right menus may be pinned—thus allowing continuous display of often-used menus deep in the menu hierarchy.

CreateGraphic, CreateAndShowGraphic

The **CreateGraphic** command is used to instantiate a graphic object. When an object is created, MapView places it in an internal list called the hold list. In order for this new object to be displayed, it must be moved to the display list by the **ShowGraphic** command. This allows the user to create many objects (off-screen) and then display them all at the same time.

If you want the object to be displayed as soon as it is created, use the **CreateAndShowGraphic** command. Each create command has the same syntax. In the example below, an object of type "AirBase" is created. The AirBase object type is defined later under the **DefineGraphicType** command. To create an instance of an AirBase object enter the following:

```
CreateGraphic airbase1 : AirBase
{
    lon = 36.50;
    lat = 24.25;
    color = "DarkBlue";
    name = "Edwards AFB";
    nRunways = 5;
    active = True;
}
```

In this example, "AirBase" is the object's graphic type and "airbase1" is the name we will give to this instance of an AirBase object. The braces "{" and "}" that enclose the block are necessary; however, the initializations inside the block are not. The user may initialize any of the object's attributes inside this block. Any attribute not initialized will get initialized automatically to the appropriate value for its type (integers and floats are initialized to 0 and 0.0000, respectively; strings are initialized to NULL; and elements of arrays are initialized according to their type).

NOTE: The **bitmap** attribute is not initialized here. Its value will be initialized to the value set in the class definition (see **DefineGraphicType**). For more information on bitmaps, see the **DefineBitmap** command.

The other attributes that *were* given default values above (**color**, **nRunways**, **name**, and **active**) are being *reset* in this **CreateGraphic** command. The positional attributes that are not set above, i.e., **x** and **y**, are set to the appropriate screen coordinate automatically.

DefineBitmap

An Icon object's graphical representation is stored in its **bitmap** attribute. Since Icon objects often share the same graphical representation, MapView stores just one copy of each unique representation (or bitmap) in a list. The **DefineBitmap** command is used to load a graphical representation from a file, associate a name with it, and store it in the bitmap list. Once a bitmap is defined, Icon objects may reference it by name. For example, to load the image of an airbase, issue the command:

```
DefineBitmap AirBaseBitmap
{
    file = "airbase.icon";
}
```

The file, airbase.icon, was created using Sun's *iconedit* program. Other icon file formats (e.g., X11 bitmap format) will be recognized in later versions of MapView. There are a number of predefined bitmaps in MapView.

Now that we have defined a suitable representation, we instantiate two Icon objects that use this bitmap:

```
CreateGraphic airbase2 : Icon
{
    bitmap = "AirBaseBitmap";
}


CreateGraphic airbase3 : Icon
{
    bitmap = "AirBaseBitmap";
}
```

Bitmaps and patterns are interchangeable. For example, you may set the pattern attribute of a Polygon object to one of the defined bitmaps, thereby tiling the polygon with copies of that bitmap image; or you may set the bitmap attribute of an Icon object to one of the defined patterns.

DefineEnumeratedType

The **DefineEnumeratedType** command is used to define an enumerated data type. Enumerated variables can only take on one of the values in its enumeration list. Unlike the C programming language, enumerations in MapView are not synonymous with integers. After a new enumerated data type is defined, variables of this type may be declared in a DefineGraphicType definition. For example, define an enumerated type **Boolean** that has two values:

```
DefineEnumeratedType Boolean { True, False }
```

DefineGraphicType

The **DefineGraphicType** command defines a new graphic type. It provides flexibility by enabling the attributes of the graphic type to be tailored to the user's needs. New graphic object types are derived from existing ones by adding attributes. At this time there is no mechanism in MapView to mask (or hide) attributes from derived base classes. To define a custom object type called "Airbase" that has three new attributes, enter the following:

```
DefineGraphicType AirBase : Icon
{
    Integer nRunways;
    String name;
    Boolean active;     #enumerated type
}
```

In this example, **DefineGraphicType** is the keyword for the command; **Icon** is the base graphic type of this new type; and **Airbase** is the name of this new object type. The ":" is best read as "derived from type". The braces "{" and "}" that enclose this block are necessary; however, the additionally defined attribute list is optional. Our new AirBase object type has three new attributes in addition to those that are inherited from the base class **Icon**. The attribute **active** is an enumerated variable as defined above under the **DefineEnumeratedType** command.

If default or initial values for any of the attributes are desired, they may be set at type-definition time, so that all instances of this type are initialized with the values provided. In object-oriented terminology, such setting represents very simple object *constructors*. An example of setting default values is shown below. Any and all of the attributes, either defined or inherited, may be initialized.

```
DefineGraphicType AirBase : Icon
{
    Integer nRunways = 2;
    String name = "default airbase";
    Boolean active = False;

    color = "LightBlue";
    bitmap = "airbase.icon";
}
```

DefinePattern

Patterns, like bitmaps, are stored in a list by MapView. Instances of the Polygon, Circle, and NetworkRegion objects use their **pattern** attribute to describe their fill pattern. The **DefinePattern** command is used to load a graphical representation for a fill pattern from a file and associate a name with it. To load a custom fill pattern from a file, issue the command:

```
DefinePattern  CrossHatch
{
    file = "crosshatch.icon";
}
```

The file, crosshatch.icon, was created using Sun's *iconedit* program. Other icon file formats (e.g., X11 bitmap format) will be recognized in later versions of MapView. There are a number of predefined patterns in MapView.

To create an object that uses our defined pattern, CrossHatch, issue the following command:

```
CreateGraphic areal : Polygon
{
    pattern = "CrossHatch";
}
```

DestroyGraphic

The **DestroyGraphic** command destroys graphic objects. If the parameter passed to the command is an object type rather than an object's name, however, all instances of this type are destroyed. For example:

```
DestroyGraphic airbasel    #destroys one instance
DestroyGraphic AirBase     #destroys all instances
```

DestroyAllGraphics

The **DestroyAllGraphics** command takes no parameters and destroys all graphic objects. The objects are removed from both the hold and display lists. Type definitions are not affected.

DestroyGraphicType

The **DestroyGraphicType** command will destroy a graphic type definition by first destroying all objects of the specified type before destroying the type itself. Use this command with caution: there is no

UnDo command. To destroy the AirBase graphic type and all instances of this type, issue the command:

```
DestroyGraphicType AirBase
```

EraseImage

The **EraseImage** command takes no parameters and is used to erase the background image and redraw it using the background color. The "background" color defaults to black, but can be changed using the "-bg" option on the command line. See Appendix A for a discussion of command-line arguments.

FrontGraphic

Use the **FrontGraphic** command to move an object to the front of the visual stack (i.e., other objects will appear behind this specified object). This command takes the name of a graphics object (or an object type) as its only parameter. See also **BackGraphic**.

LoadColormap

The **LoadColormap** command loads a new palette of colors corresponding to a CAGIS image file. This new palette may drastically affect the colors in the current image, because the colormap defines the colors to be used for each pixel value in the CAGIS image. The graphic objects, however, may or may not be affected. The objects will search the new colormap for the closest match (on hue, saturation, and value) to their **color** attribute. If a satisfactory match is not found, then the desired color will be loaded into the current colormap.

In the event that the colormap is full and no new colors may be loaded, MapView will assign the best color to the object it can. When this happens, you might notice that objects that were a rich blue appear light blue. It is for this reason that, when making CAGIS images and their corresponding colormaps, you should reserve some room for other colors. As a rule, using no more than 128 colors will usually prevent such changes. To load a new colormap, issue the command:

```
LoadColormap "lighter.cms"
```

LoadImage

The **LoadImage** command loads a Sun rasterfile or CAGIS image from a file and displays it on the screen. By convention, CAGIS image files have the extension ".image" and Sun rasterfiles have the extension ".ras". CAGIS image files require a corresponding colormap file. MapView will automatically load this file if it has the same filename as the image and the extension ".cms". Sun rasterfiles contain their own colormap information and do not require a separate colormap file. To load an image, issue the following command:

```
LoadImage "file.image"
```

MapView will load the file "image.cms" if it exists. If this file does not exist, the image will be loaded and displayed using the current or default colormap.

MoveGraphic

There are actually four distinct **MoveGraphic** commands: **MoveGraphicRel**, **MoveGraphicAbs**, **MoveGraphicLLRel**, and **MoveGraphicLLAbs**. Each of these commands takes three parameters: the name of the object, and a pair of coordinates that represent an absolute position (Abs) or a relative offset (Rel). The latter two commands (those with the LL in their name) expect their coordinate pair to be given in degrees of longitude and latitude. The other two commands take integers specifying the x and y locations, or offsets, in pixels. An example of each command is shown below:

```
MoveGraphicAbs airbase1 100 50
MoveGraphicRel airbase1  5 5
MoveGraphicLLAbs airbase1 54.6 37.8
MoveGraphicLLRel airbase1 1.0 1.0
```

A **MoveGraphic** command performs the same function on an object's position as an **UpdateGraphic** command. But if the object's position is the only attribute being changed, a **MoveGraphic** command will perform the change much faster.

Run

The **Run** command takes a double-quoted string as its only parameter. The string is not processed by MapView but is passed on to a UNIX subshell. This command, when used from within a custom menu, will execute a program or script that processes some of the current objects and returns either new objects or updates to the current ones. Since the Run command takes a quoted string as its parameter, a backslash ("\") must appear before any embedded quotes. One example of how to use the **Run** command to process some graphic objects would be the following:

```
SaveSelectedObjectsToFile objects.goal
Run "myprogram < objects.goal > newobjects.goal"
LoadFile newobjects.goal
Run "/bin/rm objects.goal newobjects.goal"
```

It is natural to see how this functionality would be very useful from within user-defined menus. For examples of how to use the **Run** command in conjunction with menus, see **CreateMenu** and **AttachMenu**.

SaveAllObjectsToFile, AppendAllObjectsToFile

The **SaveAllObjectsToFile** and **AppendAllObjectsToFile** commands are used to save the definitions of *all* objects to a GOAL file. Each takes the name of the file as its only parameter. The **Save** version will overwrite an existing file with the same name (it will prompt you first), and the **Append** version will just append the definitions of all the objects to the file specified.

```
SaveAllObjectsToFile foo.goal
AppendAllObjectsToFile foo.goal
```

SaveSelectedObjectsToFile, AppendSelectedObjectsToFile

The **SaveSelectedObjectsToFile** and **AppendSelectedObjectsToFile** commands are used to save the definitions of *selected* objects to a GOAL file. Each takes the name of the file as its only parameter. The **Save** version will overwrite an existing file with the same name (it will prompt you first), and the **Append** version will just append the definitions of the selected objects to the file specified.

```
SaveSelectedObjectsToFile foo.goal
AppendSelectedObjectsToFile foo.goal
```

SaveTypedObjectsToFile, AppendTypedObjectsToFile

The **SaveTypedObjectsToFile** and **AppendTypedObjectsToFile** commands are used to save the definitions of *all* objects of the specified *type* to a GOAL file.[4] Each takes two parameters: the name of the object type to be saved and the name of the file to write the definitions to. The **Save** version will overwrite an existing file with the same name (it will prompt you first), and the **Append** version will just append the definitions of all the objects to the file specified.

```
SaveTypedObjectsToFile Airbase foo.goal
AppendTypedObjectsToFile Airbase foo.goal
```

SelectGraphic, UnSelectGraphic

The **SelectGraphic** and **UnSelectGraphic** commands provide a means of selecting and unselecting an object by name. Selected objects are rendered with little selection boxes either around the vertices of the object (as in Polylines and Polygons) or around the bounding box of the object (as in Circles and Icons). Once selected, objects can be manipulated as a group by a number of other commands. For example:

```
SelectGraphic airbase1
UnSelectGraphic airbase1
SelectGraphic AirBase
```

Note in the third example that an object type (or class), AirBase, was specified, which will cause all objects of that type to be selected.

SetWindowBounds

The **SetWindowBounds** command is used to set the geographic information for the Map Window. The command is written out by MapView

[4]The **SaveTypedObjectsToFile** command does not currently exist. The same functionality can be achieved by using the two commands "**SelectAllGraphics** <type>" and "**SaveSelectedObjectsToFile** <file>".

as part of each of the **Save** commands and is the first command in the GOAL file. MapView uses this information when no other information about an underlying image is supplied (as is the case with all image types except CAGIS images. CAGIS images contain their geographic information within the image header). When the vector map is being used as the background image, such geographic information is necessary so that MapView will know what part of the world the Map Window was displaying at the time the objects were saved.

The syntax of the **SetWindowBounds** command is shown in three examples. The first example has a coordinate system composed of integer x's and y's that represent pixels and sets the Map Window dimensions to 800 by 600 pixels. A map projection of type "pixel" signifies that MapView is to remain in pixel mode (lats and lons are meaningless).

```
SetWindowBounds {
    projection = "pixel";
    pixel_width = 800;
    pixel_height = 600;
}
```

The second example is a **SetWindowBounds** command for a CAGIS geographic (geogrph) projection image. All CAGIS images require the corner points of the image, as well as the width and height of the image in pixels, to be specified. The projection must also be specified. A command such as this could be used to orient the Map Window to a rectangular geographic region where your data might be drawn even if you have no image to display underneath to provide a geographic reference.

```
SetWindowBounds {
    projection = "geogrph";
    pixel_width = 800;
    pixel_height = 800;
    lon_min = 47.770;
    lon_max = 48.018;
    lat_min = 29.199;
    lat_max = 29.415;
}
```

The third example demonstrates the format of the **SetWindowBounds** command for a vector map image. The coverage area of the window is specified using a center point and a nautical mile width. This is a useful way of identifying the location of the image for perspective map projections where the corner points might not be defined (as is true when you have zoomed out far enough to see the whole globe). See Appendix B for a list of all the map projections available.

```
SetWindowBounds {
    projection = "mercator";
    pixel_width = 800;
    pixel_height = 800;
    cen_lon = 46.00;
    cen_lat = 29.30;
    nautical_mile_width = 200;
}
```

ShowAllGraphics

The **ShowAllGraphics** command takes no parameters and causes all graphic objects to be displayed or refreshed. All objects on the hold list are moved to the display list, and the display list is redrawn.

ShowGraphic, UnShowGraphic

The **ShowGraphic** and **UnShowGraphic** commands take as their only parameter the name of a graphic object. The **ShowGraphic** command causes the specified object to be rendered. A side effect of this command is that the object will be displayed at the front of the visual priority (i.e., all other objects will be behind it). The **UnShowGraphic** command erases the specified object (but does not destroy it). All unshown objects are stored in an off-screen display list called the hold list. For example:

```
UnShowGraphic airbase1  # erase it
ShowGraphic airbase1    # redraw it
```

The **ShowGraphic** command is like most commands that take one parameter (e.g., an object's name) in that it may also take an object's type as a parameter. Thus, to display all objects of type AirBase, you could execute the following command:

```
ShowGraphic AirBase
```

SnapShot, SnapShotArea

The **SnapShot** and **SnapShotArea** commands are used to capture the screen (or a portion of the screen) and save it to a Sun rasterfile. **SnapShot** will save the whole screen to a rasterfile, whereas **SnapShotArea** will save only the subregion specified. These commands are very useful when you want to make a QuickTime movie (or some equivalent animation) and you would like to periodically save the current graphics screen as a "frame" in a movie. The syntax for each of these commands is demonstrated in the following example:

```
SnapShot frame_1.ras
SnapShotArea frame_1.ras 100 100 640 480
```

Here, frame_1.ras is the filename. For the **SnapShotArea** command, the four integers represent x, y, width, and height. The x and y are measured from the upper left corner of the screen (positive y-direction is downward).

UpdateAndShowGraphic

The **UpdateAndShowGraphic** command is the same as an **UpdateGraphic** command followed by a **ShowGraphic** command. The syntax is the same as for the **UpdateGraphic** command. The command is useful when you are updating relatively few objects or when you want each object's update to be reflected right away.

UpdateAndShowSelectedGraphic

The **UpdateAndShowSelectedGraphic** command updates the selected object and redraws it immediately. This command has the same syntax (inside the braces) as the two previous **Update** commands, with one exception: No object name is specified here. The object that is affected is the currently selected object (if one exists). This command is helpful when used from within a menu object (see **CreateMenu** and **AttachMenu** above) because it allows the user to select an object with the mouse and then perform a set of actions (defined by a menu item) on that object without having to mention its name. An example of the syntax for the **UpdateAndShowSelectedGraphic** command is as follows:

```
UpdateAndShowSelectedGraphic
{
    color = "SlateBlue";
}
```

UpdateGraphic

The **UpdateGraphic** command is used to update some or all of a graphic object's attributes. The object is not redrawn as a result of this command; rather, the user updates numerous objects and displays the results all at once (with a subsequent **ShowAllGraphics** command). The syntax inside the braces of this command is exactly the same as that of the **CreateGraphic** command. An example of updating the AirBase object defined above is

```
UpdateGraphic airbase1
{
    color = "Red";
    nRunways = 3;
    active = False;
}
```

UpdateGraphic Operators: +=, =+, -=, =-

Attributes, in addition to being defined, initiali d, and set, may also be incremented and decremented, and have items appended, prepended, removed from the front, or removed from the back. The operators that perform these functions are "+=", "=+", "-=", and "=-". For such scalar attributes as integers and floats, the different incrementing operators ("+=" and "=+") perform the same function. This is also true for the decrementing operators ("-=" and "=-"). For strings, however, the "+=" prepends the specified string, whereas the "=+" appends the string. The "-=" removes the desired string (if it exists) from the front of the string; the "=-" removes the string from the tail end. In the case of string lists and arrays, on the other hand, these operators affect elements at *either* the beginning or the end of the list or array. Examples of these operators are as follows:

```
UpdateGraphic airbase1
{
    color += "Light";      # prepend the word Light
    nRunways -= 1;         # decrement by one
    strlist =+ { "append this", "and this" };
    farray =- { 3.14  2.71828 };
}
```

Here, the word "Light" is prepended to whatever the color was before. On strings, the "-=" and "=-" operators will remove the first occurrence of the substring from either the left or right of the string, respectively. Also, **nRunways** is decremented by one. The **StringList** attribute, **strlist**, is having two strings appended to it. Finally, the **Float** array, **farray**, attribute is having two of its array values removed (the first occurrences from the right).

ViewFile

The **ViewFile** command takes a filename as its only parameter. The file (if it exists and is readable) will be displayed in a text window (xview editor) on the screen. This command is useful for displaying custom help files from within user-defined menus or anytime you would like to show, and allow the user to modify, the contents of a particular file. Its syntax has the following form:

```
ViewFile foo.data
```

OBJECT STRUCTURE

This section lists all the built-in graphic types in **MapView** and describes each of their attributes.

Base Types

All graphic objects are derived from one of the defined base types listed below. The base type of a graphic object determines what kind of object it is (for example, a Circle or an Icon). Each type of object has a set of attributes that specify the size, color, location, and, in some cases, the pattern or font for the object. It should be noted that all objects have attributes that permit them to be manipulated in both a pixel-based Cartesian coordinate system and a geographic latitude and longitude coordinate system. Usually, you will update an object's position in one system or the other, not both.

Pixel

The Pixel object takes up one pixel on the screen and has the following attributes:

• **color**	the color of the pixel.
• **x**	the x-coordinate in pixels
• **y**	the y-coordinate in pixels
• **lon**	degrees longitude
• **lat**	degrees latitude.

LineSegment

The LineSegment object is defined by two end points. Its attributes are defined as follows:

• **color**	the color of the line
• **width**	the line width of the line in pixels
• **x1, y1**	the x- and y-coordinates in pixels of one end of the line
• **x2, y2**	the x- and y-coordinates in pixels of the other end of the line
• **lon1, lat1**	degrees longitude and latitude of one end of the line

• **lon2, lat2**	degrees longitude and latitude of the other end of the line.

Polyline

A Polyline object is a set of connected line segments. The line width and color of the Polyline object apply to all segments in the polyline. Its attributes are defined as follows:

• **color**	the color of the polyline
• **width**	the line width of the polyline in pixels
• **points**	an array of x's and y's that specifies the vertices of the polyline
• **lonlats**	an array of longitudes and latitudes that specifies the vertices of the polyline.

Polygon

A Polygon object is described by a set of points at its vertices. It may be filled with a pattern or unfilled. The border has a variable line width; if the line width is zero, the border is not displayed. Its attributes are defined as follows:

• **color**	the color of the polygon
• **width**	the line width of the border in pixels
• **points**	an array of x's and y's that specifies the vertices of the polygon
• **lonlats**	an array of longitude and latitudes that specifies the vertices of the polygon
• **pattern**	a pattern that will be used to fill the polygon. If no pattern is specified, the polygon will be transparent. (See the **DefinePattern** command for details.)

Circle

A Circle object is described by a center point and a radius. It may be filled with a pattern or unfilled. The border has a variable line width; if the line width is zero, the border is not displayed. Its attributes are defined as follows:

• **color**	the color of the circle
• **x**	the x-coordinate of the center in pixels
• **y**	the y-coordinate of the center in pixels
• **lon**	degrees longitude of the center
• **lat**	degrees latitude of the center
• **width**	the line width of the border in pixels
• **pattern**	a pattern that will be used to fill the polygon. If no pattern is specified, the polygon will be transparent. (See the **DefinePattern** command for details.)
• **radius**	the radius of the circle in pixels
• **nautical_miles**	the radius of the circle in nautical miles.

Icon

The Icon object is represented on the screen by a bitmap that can be created using either the X11 *bitmap* program or Sun's *iconedit* program. It has the following attributes:

• **color**	the color of the icon
• **x**	the x-coordinate of the icon in pixels
• **y**	the y-coordinate of the icon in pixels
• **lon**	degrees longitude
• **lat**	degrees latitude
• **bitmap**	the name of the bitmap. (See the **DefineBitmap** command for details.)

Text

The Text object is used to display strings of text in varying fonts and colors on the screen. It has the following attributes:

• **color**	the color of the text
• **x**	the x-coordinate of the text in pixels
• **y**	the y-coordinate of the text in pixels
• **lon**	degrees longitude
• **lat**	degrees latitude
• **value**	the actual text
• **font**	the name of the font. All X11 font names are recognized.

Network

The Network object does not have its own shape. A Network is defined by the other objects that it contains. It can contain NetworkNodes, NetworkArcs, and NetworkRegions. It has the following attributes:

• **nodes**	a StringList of NetworkNode objects
• **arcs**	a StringList of NetworkArc objects
• **regions**	a StringList of NetworkRegion objects.

NetworkNode

A NetworkNode object is similar to an Icon object. The only difference is that a NetworkNode object can be used as an end point of a NetworkArc object. It has the following attributes:

• **color**	the color of the icon
• **x**	the x-coordinate of the icon in pixels
• **y**	the y-coordinate of the icon in pixels
• **lon**	degrees longitude
• **lat**	degrees latitude
• **bitmap**	the name of the bitmap. (See the **DefineBitmap** command for details.)

NetworkArc

A NetworkArc object is a line segment that spans two NetworkNodes. It has the following attributes:

• **color**	the color of the arc
• **width**	the line width of the arc in pixels
• **node1**	the first of the two NetworkNodes
• **node2**	the second of the two NetworkNodes.

NetworkRegion

A NetworkRegion object is similar to a Polygon object, except that a NetworkRegion is defined by a set of NetworkNodes at its vertices. It has the following attributes:

• **color**	the color of the region
• **width**	the width of the outline in pixels
• **nodes**	a StringList of NetworkNode objects

- **pattern** a pattern that will be used to fill the region. If no pattern is specified, the region will be transparent. (See the **DefinePattern** command for details.)

Appendix A

COMMAND-LINE ARGUMENTS

MapView's command-line arguments are listed below in its "usage" statement. The usage may also be displayed by typing "mapview -h".

Usage: **mapview [options] [debug-options]**

options:

-I path	additional directories to search for images (separated by ":"). The default is ".:./GIS:$MAPVIEWHOME:$MAPVIEWHOME/GIS:$MAPVIEWIMAGEPATH"
-G path	additional directories to search for .goal files (separated by ":"). The default is ".:./GIS:$MAPVIEWHOME:$MAPVIEWHOME/GIS:$MAPVIEWGOALPATH"
-B path	additional directories to search for bitmap files (separated by ":"). The default is ".:./GIS:$MAPVIEWHOME:$MAPVIEWHOME/GIS:$MAPVIEWBITMAPPATH"
-H directory	directory of help files (default is $MAPVIEWHELPDIR)
-cmap file	use "file" as the initial colormap file
-f font	set default font to "font"
-g file	load this GOAL file on startup
-h or -help	print this message
-i file	load this image file on startup
-ih pixels	set the Map Window height to "pixels"

-iw pixels	set the Map Window width to "pixels"
-is pixels	set the width and height to "pixels"
-pd n	set pixel distance sensitivity for object selection to n pixels (default = 50)
-pi n	set the number of commands to process between checks for a Pause request (default = 5)
-bg color	set the background color of the Map Window. Standard X color names may be used here in addition to the rrrgggbbb syntax (e.g., #00ff00 to specify green)
-fg color	set the foreground color. This option is used as the default drawing color; it is also the color objects will turn when they are selected (default is white)
-vm color	set the color of the vector map lines (default is orange)
-log file	log the MapView session to file "file" (default is $MAPVIEWHOME/lib/MapView.log)
-nolog	do not log the MapView session
-maxpoints npts	set default max array length to npts (default is 1000)
-x x_offset	x offset hint for window manager measured in pixels (default is 300)
-y y_offset	y offset hint for window manager measured in pixels (default is 0)
-exact_color	match color requests exactly. This option will use up colormap cells quickly
-close_color	if a similar color is available, use it; otherwise allocate a new color (this is the default color scheme)
-match_color	always find the closest color. This option will preserve colormap cells.

debug-options:

-sync	synchronize all packets with the X11 Server
-trace	trace execution through preselected routines
-echo	echo commands parsed on the input stream
-ml level	set malloc(3) (memory allocation) debug level (0=default, 1, or 2)
-showareas	show bounding boxes around objects.

Appendix B

MAP PROJECTIONS

The following strings may be used in the "projection =" portion of the **SetWindowBounds** command. The vector map can be rendered in each of the following map projections via the *msi* (Map Server Interface) program. The names in the left column are the ones understood by the Map Server Interface. The names in the right column are the ones that CAGIS uses to identify the projections. Either name may be used.

AlbersEqualAreaConic	albers
AzimuthalEqualArea	lambert_az
AzimuthalEquidistant	azim_eq
Equirectangular	equirect
Gnomonic	gnomonic
KavraiskyIV	
LambertConformal	lambert
LonLat	geogrph
Mercator	mercator
Miller	miller
ObliqueMercator	
Orthographic	orthogr
Perspective	gv_persp
pixel	
PolarStereographic	polrster
Polyconic	plyconic

Sinusoidal	sinusoid
Stereographic	stereogr
UTM	utm

INDEX

BIBLIOGRAPHY

Bennett, Bruce W., Hoyer, Mark, *The New RSAS Map Graphics*, RAND, Santa Monica, Calif.; MR-122-NA, October, 1992.

Zobrist, A. L., Marcelino, L. J., Daniels, G. S., *RAND's Cartographic Analysis and Geographic Information System (RAND-CAGIS): A Guide to System Use*, RAND, Santa Monica, Calif.; N-3172-RC, 1991.